ELSEVIER

Journal of Computational and Applied Mathematics 131 (2001) 473–492

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

www.elsevier.nl/locate/cam

# How WE solve PDEs

Willi Schönauer *, Torsten Adolph

*Rechenzentrum der Universität Karlsruhe, Postfach 6980, D-76128 Karlsruhe, Germany*

Received 31 August 1998; received in revised form 12 October 1999

**Abstract**

A finite difference method on an unstructured finite element mesh which we call finite difference element method (FDEM) is presented. The FDEM program package will be a black-box solver for nonlinear systems of elliptic and parabolic PDEs with mesh refinement and automatic control of the consistency order in each space grid point. In this paper we present the solution method (with examples) for 2-D systems of elliptic PDEs. © 2001 Elsevier Science B.V. All rights reserved.

*MSC:* 65N06; 65G99

*Keywords:* Finite difference method; Unstructured grid; Nonlinear PDEs; Mesh refinement; Order control

## 1. Introduction

We want to design a black-box solver for nonlinear systems of elliptic and parabolic PDEs. This means that the user of the corresponding program package enters his PDEs and boundary conditions (BCs) and the corresponding Jacobian matrices as Fortran code together with an initial mesh into some given program frames and the black-box tries to find the solution with a prescribed relative error.

Such a black-box solver must be extremely robust. We want the following properties: Full flexibility of the

- PDE operator: all types of nonlinear elliptic and parabolic systems of PDEs,
- BC operator: all types of nonlinear BCs,
- solution operator: optimal self-adapted consistency order, error estimate, mesh refinement,
- geometry: arbitrary domain with unstructured (initial) mesh.

* Corresponding author.
*E-mail address:* schonauer@rz.uni-karlsruhe.de (W. Schönauer).


PII: S0377-0427(00)00255-7

This is a very ambitious goal. We were able to realize this goal only by the experience that we had gained over many years, gradually generalizing the FDM to more and more geometrical flexibility. We call this method now FDEM: finite difference element method, it uses the FDM on a typical unstructured FEM mesh. This is obtained by quite elementary building blocks as will be seen below.

One may ask: why not use the FEM? We have designed in our group the VECFEM program package [7] as a black box solver for general functional equations as they result from the weak formulation of PDEs. We have seen from this valuable experience that an engineer who wants to solve a PDE has to reformulate his problem as a functional equation (with difficulties for certain BCs), that it is difficult to get an error estimate for such a general case [8] that the consistency order is de facto limited by the properties of the mesh generator and that it is under these conditions impossible to use a local order control.

There are several other program packages available with a similar goal, but none has all the properties requested above for FDEM. The vectorized local uniform grid refinement (VLUGR) code [4–6] is a PDE black-box solver of CWI, Amsterdam. It is a FDM of second order to solve parabolic equations with the method of lines. It has been designed especially for solutions with steep gradients, e.g. flame fronts, where the refined grid follows the activity zone. The domain can be any area that can be described by right-angled polygons (2-D) and polyhedrons (3-D). There is no variable consistency order in space and no error estimate.

Another approach to generate difference formulae of arbitrary consistency order on an unstructured grid is the essentially nonoscillatory (ENO) approach of Abgrall [1]. This is a finite volume method (FVM) where difference formulae of theoretically arbitrary consistency order can be generated on unstructured 2-D grids, but practically only up to order 3 are used. The method is specially designed for compressible flow calculations. The interesting point is that the grid points that generate the difference formulae are selected so that nonoscillatory solutions are obtained. However, near the boundary the order must be reduced. There is no black-box solver with error estimates and self-adaptation on the basis of ENO schemes.

A widely used tool for numerical experiments with PDEs is the piecewise linear triangles multi-grid (PLTMG) program package [2]. Only the User's Guide 7.0 was available to us. It solves a single (scalar) nonlinear elliptic PDE on a mesh of linear triangles by a linear FEM on a 2-D domain. PLTMG has a grid generator, refines adaptively the mesh and bases a MG preconditioning for a CG solver on a hierarchical sequence of meshes. The strength of PLTMG is its ability to detect and treat singular points of the solution, however, it cannot treat systems of PDEs.

Another quite ambitious program package is unstructured grid (UG) of the group of Wittum [3]. The goal is the development of a parallel software platform for the efficient solution of PDEs by different FEMs and FVMs. It includes adaptive grid refinement and coarsening and uses MG for the solution of the large linear systems. A library of finite elements of different types, including mortar elements is included. However, like usual for FEM packages for each type of PDE a specialized version must be generated so that UG may be called an "indirect" black-box solver in the form of a tool box. Error estimates are not included as standard tool.

Rannacher [11] develops an efficient a posteriori error estimate for the Galerkin FEM by the use of the adjoint problem that is used to refine the mesh. He demonstrates the application by several examples. However, neither an attempt to design a black-box solver is made nor the use of higher order methods is considered.

There are many other program packages to solve PDEs, mostly designed for a special problem. However, to our knowledge there exists no code that fulfils all the requirements that we have formulated for an "ideal" PDE black-box solver. Our FDEM program package that is still under development has the ambitious goal to fulfil these requirements. In the present paper the (single processor) algorithm and code for 2-D is discussed. The next steps are the (single processor) 3-D code and then the parallelization for distributed memory parallel computers.

The organization of this paper is as follows: In Section 2 the error equation is presented that allows a transparent discussion of the influence of all error sources and a balancing of the different error types. In Section 3 we discuss the algorithm to select for each node an individual optimal consistency order. In Section 4 the algorithm for the mesh refinement is presented so that a requested relative tolerance is met. The access to the discretization error which is the innermost kernel of the solution method is presented in Section 5. It is based on families of difference formulae on the unstructured grid and it is discussed, how such formulae are generated. In Section 6 an example is presented with fixed order and mesh, with pure order control, with pure mesh refinement and with selfadapted order and mesh. In Section 9 some remarks to the linear solver LINSOL are made. This solver is used for the examples of this paper, but it is not discussed in detail.

## 2. The error equation

The error equation is the basis of our solution method. It has been developed in its basic form for the FIDISOL program package (see [13, Section 17]). It is discussed here for the present 2-D PDE operator. We abbreviate the PDE and BC operator for a solution $u(x, y)$ as follows:

$$Pu \equiv P(t, x, y, u, u_t, u_x, u_y, u_{xx}, u_{yy}, u_{xy}) = 0. \tag{1}$$

If we include $t, u_t$, we have a parabolic problem. Here we discuss only elliptic problems, without $t, u_t$. For the solution of parabolic problems we will use the solution method with order and step size control in the $t$-direction as in FIDISOL. $Pu$ is an arbitrary nonlinear function of its arguments. Therefore, we use a Newton–Raphson approach

$$u \Longleftarrow u^{(\mathrm{v}+1)} = u^{(\mathrm{v})} + \Delta u^{(\mathrm{v})} \tag{2}$$

and we drop immediately the iteration index v and we linearize (1) in the Newton correction *function* $\Delta u$, which means that we get also the corresponding derivatives of $\Delta u$, e.g., $\Delta u_{xx}$. We then get a linear PDE for the Newton corrrection function:

$$\begin{aligned} Q\Delta u \equiv & -\frac{\partial P}{\partial u}\Delta u - \frac{\partial P}{\partial u_t}\Delta u_t - \frac{\partial P}{\partial u_x}\Delta u_x - \cdots - \frac{\partial P}{\partial u_{xy}}\Delta u_{xy} \\ & = P(t, x, y, u, u_t, u_x, u_y, u_{xx}, u_{yy}, u_{xy}). \end{aligned} \tag{3}$$

$Q\Delta u$ is now a linear PDE operator for $\Delta u$, $P(\ldots) \equiv Pu \equiv Pu^{(\mathrm{v})}$ is the Newton residual or Newton defect $\neq 0$. If $u^{(\mathrm{v})}$ were the exact solution $u$ we would have $Pu = 0$.

We now discretize (index d) the PDE (3) by replacing function values by their value on a grid and derivatives by difference formulae

$$\Delta u \Longleftarrow \Delta u_{\mathrm{d}}, \quad \Delta u_t \Longleftarrow \Delta u_{t,\mathrm{d}}, \quad \Delta u_x \Longleftarrow \Delta u_{x,\mathrm{d}}, \ldots, \tag{4}$$

where, e.g., $\Delta_{x,\mathrm{d}}$ is the difference formula for $\Delta u_x$. However, the derivatives of the function $u = u^{(\mathrm{v})}$ in $P(\ldots)$ are replaced by difference formulae plus their error estimates:

$$u \Longleftarrow u_{\mathrm{d}}, \quad u_t \Longleftarrow u_{t,\mathrm{d}} + d_t, \quad u_x \Longleftarrow u_{x,\mathrm{d}} + d_x, \ldots, u_{xy} \Longleftarrow u_{xy,\mathrm{d}} + d_{xy}, \tag{5}$$

where, e.g., $d_x$ is the discretization error estimate for the difference formula $u_{x,\mathrm{d}}$. In (4) we do not use error estimates because these would be errors of errors that go to zero with the Newton correction. We now linearize also in the discretization errors $d_\mu$ (we will see below how we compute the $d_\mu$). If we order the coefficients of the unknown vector $\Delta u_{\mathrm{d}}$ that result from this discretization of (3) in a matrix $Q_{\mathrm{d}}$ we can formally express the new error as

$$\begin{aligned} \Delta u_{\mathrm{d}} &= \Delta u_{Pu} + \Delta u_{D_t} + \Delta u_{D_x} + \Delta u_{D_y} + \Delta u_{D_{xy}} \\ &= Q_{\mathrm{d}}^{-1}((Pu)_{\mathrm{d}} + D_t + \{D_x + D_y + D_{xy}\}). \end{aligned} \tag{6}$$

*level of solution* (under $\Delta u_{Pu}$)

*level of equation* (under $(Pu)_{\mathrm{d}}$)

This is the error equation and the key to the solution process. Quite naturally $Q_{\mathrm{d}}$ is a large and sparse matrix and we do not explicitly compute $Q_{\mathrm{d}}^{-1}$ but solve iteratively the corresponding linear system. In the parentheses of the second row of (6) we have errors "on the level of the equation" or on the consistency level that are transformed by $Q_{\mathrm{d}}^{-1}$ to the level of the solution. $(Pu)_{\mathrm{d}}$ is the discretized Newton residual, i.e., it is the operator $P(\ldots)$ in (1) where derivatives have been replaced by difference formulae that are evaluated for $u_{\mathrm{d}} = u_{\mathrm{d}}^{(\mathrm{v})}$. The $D_\mu$ are discretization error terms that result from the linearization in the $d_\mu$, e.g.,

$$D_x = \frac{\partial P}{\partial u_x} d_x + \frac{\partial P}{\partial u_{xx}} d_{xx}. \tag{7}$$

In the first row of (6) we have errors "on the level of the solution". The overall error $\Delta u_{\mathrm{d}}$ has been split up into the parts that result from the corresponding terms on the level of the equation. The Newton correction is

$$\Delta u_{Pu} = Q_{\mathrm{d}}^{-1}(Pu)_{\mathrm{d}} \quad \text{or} \quad Q_{\mathrm{d}} \Delta u_{Pu} = (Pu)_{\mathrm{d}}, \tag{8}$$

where we use the second formulation for the computation by iterative solution. The Newton correction $\Delta u_{Pu}$ is the only error that is applied according to (2) to the solution. The other (discretization) errors are only used for the error control (if we applied them, we had no more an error estimate).

In the braces $\{\}$ in (6) we have the space key error which is given for a certain mesh, consistency order and actual solution. In the sense of error balancing we adapt the Newton residual $(Pu)_{\mathrm{d}}$ to $\{\}$ by stopping the Newton–Raphson iteration if $(Pu)_{\mathrm{d}}$ is small by a safety factor compared to $\{\}$. For the solution of parabolic PDEs we adapt $D_t$ to $\{\}$ by a corresponding choice of the time step $\Delta t$, see [13].

Up to now we have explained the error equation as if we had a scalar PDE. However, all considerations hold likewise if we have a system of $m$ PDEs. Then we have

$$u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{pmatrix}, \quad Pu = \begin{pmatrix} P_1 u \\ P_2 u \\ \vdots \\ P_m u \end{pmatrix}, \quad \frac{\partial P}{\partial u_x} = \begin{pmatrix} \dfrac{\partial P_1 u}{\partial u_{1,x}}, \ldots, \dfrac{\partial P_1 u}{\partial u_{m,x}} \\ \vdots \\ \dfrac{\partial P_m u}{\partial u_{1,x}}, \ldots, \dfrac{\partial P_m u}{\partial u_{m,x}} \end{pmatrix} \tag{9}$$

and likewise for the other Jacobian matrices. The description is more complicated, but the method is the same.

In the following, we need different types of norms. We always use max norms. The index d means "discretized", i.e., the solution on the difference grid, $l$ denotes the component in a system of $m$ PDEs with $m$ solution components, $i$ denotes the global node number. The global norm is

$$\|u_\mathrm{d}\| = \max_{\substack{l=1,m \\ \text{nodes } i}} |u_{\mathrm{d},l,i}|. \tag{10}$$

A local norm is

$$\|u_{\mathrm{d},i}\| = \max_{l=1,m} |u_{\mathrm{d},l,i}|. \tag{11}$$

For the errors we use global relative norms. The global relative norm of the error of a component $l$ is

$$\|\Delta u_{\mathrm{d},l}\|_\mathrm{rel} = \frac{\|\Delta u_{\mathrm{d},l}\|}{\|u_{\mathrm{d},l}\|}, \tag{12}$$

a local relative error of type $\|\Delta u_{\mathrm{d},l}/u_{\mathrm{d},l}\|$ is not useful because $u_{\mathrm{d},l}$ may go through zero. The global relative norm of the error of the solution is

$$\|\Delta u_\mathrm{d}\|_\mathrm{rel} = \max_{l=1,m} \|\Delta u_{\mathrm{d},l}\|_\mathrm{rel}. \tag{13}$$

This error then will be checked against a prescribed relative tolerance tol, see below.

## 3. Choice of the consistency order $q$

We denote the consistency order for the space discretization (elliptic case) by $q$. A basic solution is computed with the prescribed initial order $q = 2$ or 4 (or 6, this order is already critical). We compute for this basic solution in each node $i$ the local space key error (see (6)) norm for the orders $q = 2, 4, 6$:

$$\|\{D_x + D_y + D_{xy}\}_i\|_{q=2,q=4,q=6}, \tag{14}$$

i.e., we compute this norm using difference formulae of orders $q = 2, 4, 6$ (independent of the initial order). We want to select the "best" order. Naively one would select the order with the smallest norm. However, higher order is more expensive because higher order difference formulae have more

nodes, thus create more nonzero coefficients in the global matrix and the matrix–vector multiplication in each iteration step is more expensive. Therefore, we accept a higher order only if

$$\|\{\}_i\|_{\text{higher order}} \leqslant f \|\{\}_i\|_{\text{lower order}} \tag{15}$$

with $f$ as a "tuning parameter" that is chosen to minimize the overall computation time. We will meet in the following still further such tuning parameters which result from "numerical engineering" considerations. Here, we should explain the notion "numerical engineering" which has been created as far as we know by Hans Stetter, Vienna (although no direct reference is known to us). Engineering means that practical experience is involved and compromises must be made. By introducing empirical parameters the solution process can be speeded up. A well-known example is the relaxation factor that speeds up the Gauss–Seidel method to the SOR or SSOR method. For general matrices this relaxation factor must be determined empirically, this is numerical engineering. Now we go back to Eq. (15): For the example that is presented below we selected between the orders 4 and 2, $f = 0.7$ and between the orders 6 and 4, $f = 0.01$ because the order 6 is rather critical. This means that the error norm of order 6 must be below 1% of the error norm of order 4 for us to switch to the order 6.

Finally, we have by this method an *individual order* $q_i$ in each node $i$. This is unique in the development of a FDM for elliptic problems on an unstructured grid.

## 4. Mesh refinement

The user prescribes a global relative error tol that is checked against (13) so that finally

$$\|\Delta u_{\mathrm{d}}\|_{\mathrm{rel}} \leqslant \mathrm{tol}. \tag{16}$$

The computation starts with an initial mesh, presently we use linear triangles, but it would be no problem to use linear quadrilaterals. The requested accuracy can be obtained only by mesh refinement. The relative tolerance tol is given on the level of the solution in the sense of Eq. (6). However, for the control of the mesh size that is made on the level of the equation we need a corresponding value tolg. How can we transform tol to tolg? We have from (6) and from the numerical computation the information, how the Newton correction is related to the Newton residual:

$$\Delta u_{Pu} = Q_{\mathrm{d}}^{-1}(Pu)_{\mathrm{d}} \tag{17}$$

from which we define a "norm transforming factor"

$$\|Q_{\mathrm{d}}^{-1}\| := \frac{\|\Delta u_{Pu}\|}{\|(Pu)_{\mathrm{d}}\|} \tag{18}$$

for the transformation from the level of the equation to the level of the solution. Now we make an analogy consideration:

$$\text{"tol"} \left\{ \frac{\| \Delta u_{Pu} \|}{\| u_{\mathrm{d}} \|} = \| Q_{\mathrm{d}}^{-1} \| \overbrace{\frac{\| (Pu)_{\mathrm{d}} \|}{\| u_{\mathrm{d}} \|}}^{\text{"tolg"}} \right. , \tag{19}$$

where the l.h.s. has the character of tol (relative error on the level of the solution) and $\|(Pu)_{\mathrm{d}}\|$ has the character of tolg (error on the level of the equation). So we get the approach

$$\mathrm{tolg} := \mathrm{tol}\|u_{\mathrm{d}}\| \frac{\|(Pu)_{\mathrm{d}}\|}{\|\Delta u_{Pu}\|}. \tag{20}$$

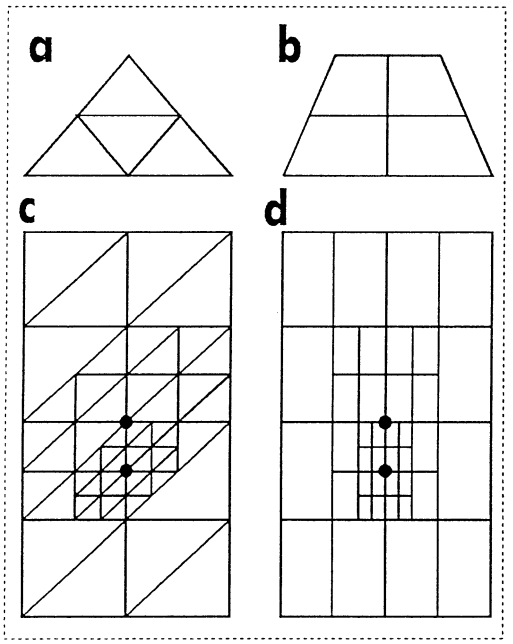


Fig. 1. (a,b) refinement of a linear triangular and quadrilateral element, (c,d) refinement of a mesh where the upper black dot is a first and the lower a second refinement point.

This is again typical numerical engineering. It is clearly a "coarse" approach but we can "correct" it by a corresponding "safety factor".

Now we check at each node $i$ if

$$\|\{D_x + D_y + D_{xy}\}_i\| \leqslant s_s \,\mathrm{tolg}. \tag{21}$$

Here $s_s$ is again a tuning parameter (or safety factor) that must be determined to minimize the solution time. In the example presented below we use $s_s = 24$ which shows that (20) is a rather coarse approach but one which is not easily improved as with the computationally available data.

A node where check (21) fails is a refinement point. If in an element at least one of the vertices is a refinement point, the whole element will be refined, by halving the edges, see Fig. 1(a) and (b). In the meshes Fig. 1(c) and (d) the upper black dot is a refinement point in a first cycle and the lower black dot a refinement point in a second cycle. So locally refined meshes result. One of the new 4 elements gets the number of the old element and 3 new element numbers are generated. In some cases the intermediate node on an edge may already be present from the refinement of a neighboring element. If it is not yet existing, a new node will be generated. The function value for a newly created node is determined in the following way: We interpolate from both end points of the subdivided edge the function value for the mid-point, using the interpolation formula (29), see below, of consistency order $q_i$ of the end point $i$. Then we take the mean value of the interpolated values. The order of the new node is the min of the orders of the generating vertices.

For the basic organization of the element lists we admit on an edge only 3 nodes. If by the refinement process on an edge a 4th node would be created which is the case if a triangle of size "1" has neighbors of size "$\frac{1}{4}$" as can be seen in Fig. 1(c) at the right triangle in the second row from the bottom, this triangle must also be refined, but not because of the error but because of the list organization. This has an consequence that the refinement process "propagates" in the environment. This results in a rather complicated program logic.

After the determination of the new mesh a new solution at the new orders of accuracy is computed. The determination of the order is made on the "old" grid, i.e., before the refinement. Finally, we check the accuracy requirement (16) and if it is true the computation is finished. If it is false, we start a next cycle with the determination of new orders and a new mesh until (16) is true or a maximal prescribed number of cycles has been attained.

## 5. The estimate of the discretization error and the generation of the difference formulae

In the error equation (6) we assume that we have estimates for the discretization errors which are visible in (7) where we need $d_x$, $d_{xx}$, the estimates for the discretization errors of the difference formulae $u_{x,\mathrm{d}}$ and $u_{xx,\mathrm{d}}$, see (5). We explain the procedure for the derivative $u_x$:

$$u_x = u_{x,d,q} + \bar{d}_{x,q} = u_{x,d,q+2} + \bar{d}_{x,q+2}, \tag{22}$$

where the index $q$ and $q+2$ denotes the consistency order of the formula and $\bar{d}_{x,q}$, $\bar{d}_{x,q+2}$ denote the exact discretization errors for the formulae of order $q$ and $q+2$. If we resolve the second and third part of (22) for the error of the actual order $q$ and neglect the error of the higher order formula we get the estimate

$$d_x := u_{x,d,q+2} - u_{x,d,q} \quad \{+ d_{x,q+2}\}. \tag{23}$$

In the braces we have indicated the neglected term. If we resolve the first and second part of (22) for the exact error $\bar{d}_{x,q} = \bar{d}_x$, we get

$$\bar{d}_x = u_x - u_{x,d,q} \tag{24}$$

which means that we have replaced in (23) the derivative $u_x$ by a higher order difference formula. This shows you that one should apply approach (23) only with utmost care: you must be sure that the (exact) error decreases with increasing order, i.e., the neglected error in (23) must be smaller than the estimated error. If you apply (23) without such a control, the estimate may fail. This is a disagreable experience that we learned soon when we used this approach in the FIDISOL program package, see [13, Section 17]. Another experience we have made is the following: Happy are those people who do not see the errors. Since we can "see" the errors, we are often very unhappy.

At this point it is appropriate to discuss our error approach in more detail. We operate flexibly with variable consistency order. The naive notion would be: the higher the order, the better it is. However, higher order formulae have more grid points and thus extend farther into the surrounding grid, introducing false information, e.g., if the function values change rapidly. Therefore on a coarse grid higher order formulae are worse than lower order formulae. High-order formulae pay only if we have a high-accuracy requirement which is quite naturally coupled with a fine mesh. The best we can do is to make the program "intelligent" so that it determines the individual optimal order for each grid point itself. To our knowledge we are the first to realize this in a general black-box solver.

This is quite different from the $p$-version of FEM codes. Our choice of individual order would mean in the FEM-world that each element may have a different order from its neighbor element. In our FDEM method we have the necessary information to determine the optimal order for each node. So even in a single triangle the order of the three nodes may be different, e.g., 2 for the first node, 4 for the second node and 6 for the third node. For our solution method it seems that the order $q = 4$ is the best for many examples, see later the results of the selfadaption process. So, if one wants to operate with fixed order, $q = 4$ is recommended. We also found that order 8 that needs order 10 for the error estimate, is rather critical. Also odd orders do not give better results than the preceding even orders. Therefore, we restricted the order control to the orders $q = 2, 4, 6$ for practical reasons. A few changes in the code would allow arbitrary orders.

Another point of our solution method is the complete transparency of the error propagation. Errors are created by the replacement of derivatives by difference formulae which causes the exact discretization errors $\bar{d}_\mu$ that are made "computable" by approach (23) as error estimates $d_\mu$, and by the stopping of the Newton iteration that causes the defect $(Pu)_\mathrm{d}$. Eq. (8) shows immediately, how from the Newton residual $(Pu)_\mathrm{d}$ results the Newton correction $\Delta u_{Pu}$. The propagation of the discretization errors $d_\mu$ into a contribution to the overall error is more complicated. This can be seen for the $x$-discretization errors in Eq. (7). The errors $d_x$, $d_{xx}$ are multiplied by the corresponding Jacobian matrices and then the resulting $D_x$ is multiplied by $Q_\mathrm{d}^{-1}$. Nevertheless, we can follow explicitly the "way" of the errors. This is the case because in the FDM we use immediately the PDEs and we get strong solutions.

In the FEM one would call our procedure roughly a $p$-method (or in our notation one should rather say a $q$-method). However, in the FEM a transparent following of the errors caused by the shape function approach is not possible because instead of the PDE a functional equation that is obtained by a weighted mean is solved. Here the "discretization" process is completely integrated into the whole solution process. As a consequence a weak solution is obtained. Grosz [8] has investigated how the ideas of our procedure can be carried over to a FEM black-box solver. He pointed out how to check the error of an order $q$ by the order $2q$ which gives immediately problems for higher order (shape functions). To develop a similar FEM with variable self-adapted order, in this case for each element, will not be possible.

The next problem is, how to generate difference formulae of arbitrary consistency order $q$ on an unstructured mesh. Here, we use the approach that we have developed in the CADSOL program package [12] for a body-oriented grid. For 2-D we make a polynomial approach in $x, y$ of order $q$ which then will be the consistency order, i.e. the generated formulae are exact for polynomials of order $q$:

$$P_q(x, y) = a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 x y + a_5 y^2 + \cdots + a_{m-1} y^q. \tag{25}$$

We need $m = (q + 1)(q + 2)/2$ grid points or nodes to determine the $m$ unknown coefficients $a_0$ to $a_{m-1}$. However, form (25) is not suited for the generation of difference formulae where we need explicitly the function values multiplied with certain coefficients. Therefore, we introduce the *influence polynomials* $P_{q,i}$ for the nodes $i = 0, m - 1$

$$P_{q,i}(x, y) = \begin{cases} 1 & \text{in node } i, \\ 0 & \text{in other nodes.} \end{cases} \tag{26}$$

The influence polynomials are a complete analogon to the shape functions in the FEM, but used in a quite different context.

For the determination of the $P_{q,i}\,(x,y)$ we put the coordinates $x_i, y_i$ of the selected $m$ nodes into approach (25) which yields a linear system with $m$ right-hand sides:

$$\begin{array}{lllll} \text{equ.,} & & & i=0\ \ldots\ m-1, \\ 0: & 1a_{0,i} & +x_0a_{1,i} \quad +\cdots+\ y_0^q a_{m-1,i} & =1 & 0, \\ & \vdots & & 0 & \vdots \\ & & & \vdots & 0, \\ m-1: & 1a_{0,i} & +x_{m-1}a_{1,i} \ +\cdots+\ y_{m-1}^q a_{m-1,i} & =0 & 1. \end{array} \tag{27}$$

Each solution vector $a_{0,i}$ to $a_{m-1,i}$ represents the set of coefficients for the influence polynomial $P_{q,i}(x,y)$. If we denote the matrix of these linear systems by $M$ and the matrix whose columns are the solution vectors of (27), i.e., the coefficients of the influence polynomials, by $A$ we get

$$MA = I, \quad A = M^{-1}. \tag{28}$$

This yields an interesting interpretation: the columns of the inverse $M^{-1}$ of $M$ are the coefficients of the influence polynomials. This property results exactly from the definition of the influence polynomials.

With the influence polynomials we have immediately an interpolating polynomial for the $m$ function values $u_i$ of the $m$ nodes:

$$u_{\mathrm{d}}(x,y) := \sum_{i=0}^{m-1} u_i P_{q,i}(x,y). \tag{29}$$

Thus, we have a functional expression for the discretized function values $u_{\mathrm{d}}$. Evaluation of $P_{q,i}(x_j, y_j)$ give the coefficients of an interpolation formula for the node $j$.

Derivatives of (29) yield the desired difference formulae, e.g.,

$$u_{x,\mathrm{d}} := \frac{\partial u_{\mathrm{d}}(x,y)}{\partial x} = \sum_{i=0}^{m-1} u_i \frac{\partial P_{q,i}(x,y)}{\partial x} \tag{30}$$

and similarly for the other derivatives. The evaluation of the derivatives for a node $j$ that is the node where the formula is needed yields the coefficients of the difference formula. For the error estimates like (23) direct error formulae can be generated. So we have from (30) also explicit expressions for the error estimates $d_\mu$.

The problem is now – and this is the key to FDEM – how to determine $m$ appropriate nodes on an *unstructured mesh* so that (27) delivers "good" coefficients for the influence polynomials. We needed several years of continued research to get the final solution of this problem that is presented below.

The mesh is generated by a (commercial) mesh generator, e.g., IDEAS or PATRAN. The basic information is the nek-list that gives for each element the global node numbers of its nodes, see Fig. 2 for a mesh of linear triangles, $k$ is the local node number of the triangle with number $e$. It is easy to invert such a list so that we have for each node the element numbers in which this node appears. Fig. 3 shows a linear triangular mesh of 751 nodes and 1410 elements generated by the IDEAS mesh generator on the unit circle. Now we search for each node its neighbors in "rings" of

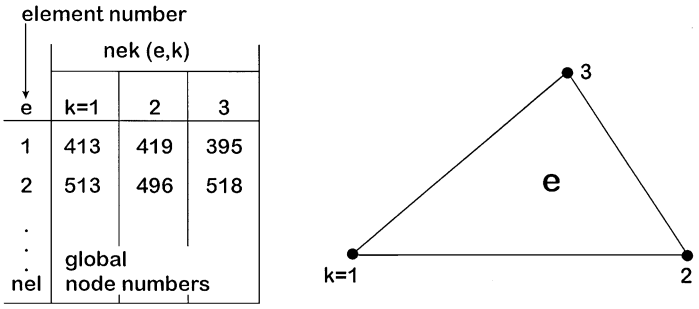


Fig. 2. nek-list of global nodes for a mesh of linear triangles.

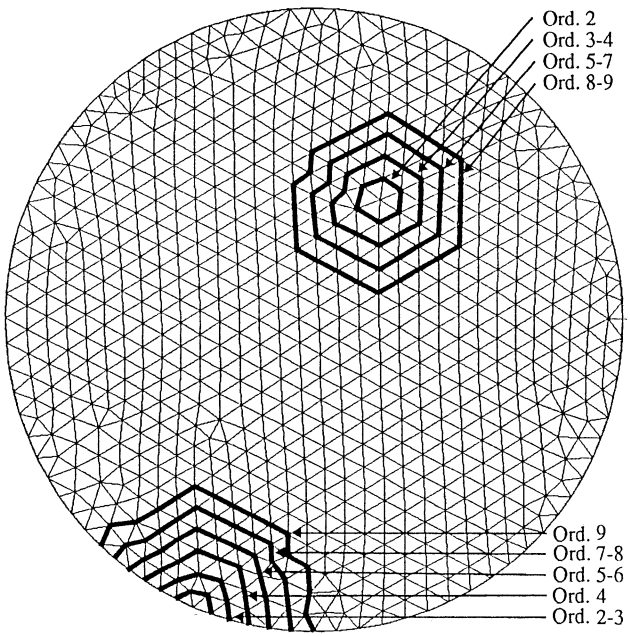


Fig. 3. Linear triangular mesh with 751 nodes and 1410 elements generated by the IDEAS mesh generator on the unit circle.

elements by the inverted nek-list and by a mask that excludes already used nodes. The procedure has been presented in detail in [15]. There are presented tables for the derivatives and the result of the difference formulae, for the exact and the estimated errors. Usually designers of difference formulae ask: how good is my computed derivative. We ask: how good is my error estimate and this is a quite different level of quality. Schönauer [15] deals only with the generation of difference formulae, not with the solution of PDEs. The application of these formulae to the solution of PDEs is presented in this paper.

Because the linear systems (27) are very "critical", we do not search only for so many rings that we get the requested $m$ nodes for the order $q$, but we search up to an order $q + \Delta q$. This means

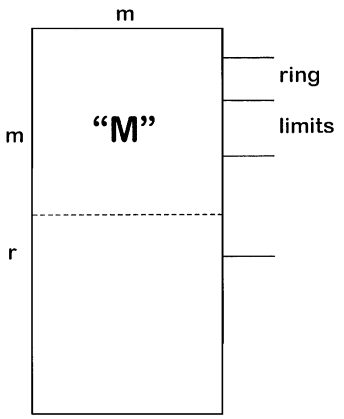


Fig. 4. Illustration for the computation of the coefficients of the influence polynomials.

that we have for our $m$ unknown coefficients $a_j$ of the influence polynomials more than $m$ equations, namely $m+r$ equations available, see Fig. 4. The reason is that in $m$ nodes usually there is not sufficient information for the $m$ coefficients, e.g., if the nodes are on straight lines as can be seen in Fig. 3. Then the matrix $M$ in (28) would be singular.

The $m+r$ nodes are transformed to the square between $-1$ and $+1$ in $x$- and $y$-direction to get "normalized" equations and the corresponding equations are arranged according to the generating rings, see Figs. 3 and 4. We execute the Gauss–Jordan algorithm for the computation of the inverse $M^{-1}$ (28) with *row pivoting*. The pivoting would select the row (node) with largest pivot which is usually an equation "far below" in the system that corresponds to a node far away on an outer ring. However, for difference formulae we want nodes near the center node to get "good" formulae. Therefore, we allow a crossing of a ring limit only if $|\text{pivot}| < \varepsilon_{\text{pivot}}$, where $\varepsilon_{\text{pivot}}$ is a given threshold value. The values $\Delta q$ for the ring search and $\varepsilon_{\text{pivot}}$ for the Gauss–Jordan algorithm have a *decisive* influence on the quality of the difference formulae. The following values have been found to deliver the best formulae for the consistency order $q$:

$$\begin{array}{llll} q & \Delta q & \varepsilon_{\text{pivot}} \\ 2 & 4 & 10^{-2} \\ 4 & 6 & 5\cdot 10^{-3} \\ 6 & 8 & 10^{-3} \end{array} \tag{31}$$

This is again typical numerical engineering. It must be noticed that we need for the consistency order $q=8$ the order $q+2=10$ for the error estimate. Our code is designed for arbitrary consistency order, but it turned out that the order $q=10$ is already so critical that most examples failed and where they succeeded the results were not better than for order $q=8$. So we limited our program by the order 6 and investigated only the orders given in (31). It should be mentioned that we generate the error formulae directly from the same system of equations of Fig. 4 so that $\Delta q \geqslant 2$ must hold. The equations for the error formulae are "extensions" of the equations for the difference formulae.

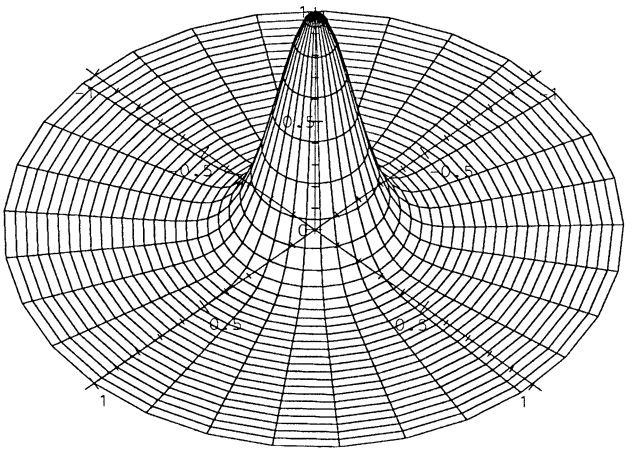

Fig. 5. Test function $\bar{u}$ (34) on the unit circle.

## 6. Examples

We want to solve the system of 3 PDEs

$$\begin{aligned} &u_{xx} + u_{yy} + \omega_y - f_1 = 0, \\ &v_{xx} + v_{yy} - \omega_x - f_2 = 0, \\ &u\omega_x + v\omega_y - (\omega_{xx} + \omega_{yy}) - f_3 = 0, \end{aligned} \tag{32}$$

under the BCs

$$u - g_1 = 0, \quad v - g_2 = 0, \quad \omega + u_y - v_x - g_3 = 0. \tag{33}$$

This is a model for the 2-D Navier–Stokes equations with Reynolds number $\mathrm{Re} = 1$ in velocity-vorticity form. We have added terms $f_i$ and $g_i$ that are determined that $u = v = \omega = \bar{u}(x, y)$ is a given solution.

We use as basic mesh the mesh of Fig. 3 with 751 nodes, 1410 elements on the unit circle and as test function

$$\bar{u}(x, y) = \mathrm{e}^{-32(x^2+y^2)} = \bar{v}(x, y) = \bar{\omega}(x, y). \tag{34}$$

This is a sugar-loaf with $\bar{u} = 1$ in the center of the unit circle and rapid decay to the boundaries, see Fig. 5. We use as starting solution for the Newton–Raphson iteration $\bar{u}$, $\bar{v}$, $\bar{\omega}$, but if we use $u = v = \omega = 1$ we observed even faster convergence. The computer is one processor of the Fujitsu VPP300, a vector processor with 2.24 GFLOPS theoretical peak performance, 2 GB memory and a half-performance length $n_{1/2} = 300$ which means that short vector length is quite inefficient, see Chapter 5 in [16]. We use for the iterative solution of the resulting large and sparse linear system our linear solver LINSOL [14] and there we used the BICO method, a smoothed biconjugate gradient method, see [13].

In Table 1 we present the results of the error estimates for different consistency orders $q$. For $q = 2$ we give the global relative component errors (12) for the three components $u, v, \omega$. The maximum of these component errors is the global relative error (13) which in this case and in most of the other

Table 1
Global relative errors for different consistency orders $q$. The solution time is $t$ in CPU-s

| $q$ | $t$ (s) | Component | Estimated | Exact |
|---|---|---|---|---|
| 2 | 5.7 | $u$ | $0.392 \cdot 10^{-1}$ | $0.396 \cdot 10^{-1}$ |
| | | $v$ | $0.398 \cdot 10^{-1}$ | $0.437 \cdot 10^{-1}$ |
| | | $\omega$ | $0.482 \cdot 10^{-1}$ | $0.490 \cdot 10^{-1}$ |
| 4 | 8.2 | $(\omega)$ | $0.098 \cdot 10^{-1}$ | $0.148 \cdot 10^{-1}$ |
| 6 | 30.1 | $(\omega)$ | $0.184 \cdot 10^{-1}$ | $0.038 \cdot 10^{-1}$ |

Table 2
Global relative errors for the second cycle with pure order control

| Order of basic solution | Time 2nd cycle (s) | No. of nodes with order $q=$ | | | Global relat. error | |
|---|---|---|---|---|---|---|
| | | 2 | 4 | 6 | Estimated | Exact |
| 2 | 10.8 | 210 | 541 | 0 | $0.085 \cdot 10^{-1}$ | $0.160 \cdot 10^{-1}$ |
| 4 | 10.8 | 207 | 541 | 3 | $0.087 \cdot 10^{-1}$ | $0.149 \cdot 10^{-1}$ |

cases is that for the component $\omega$. The error estimate, compared to the exact error, is excellent. It should be mentioned that we do not compare the quality of the solution but that of the quality control which is one level better than usual. These error estimates are the maxima over the 751 nodes! For $q = 4$ the errors go down, the error is underestimated by a factor of 1.5, again the maximum over 751 nodes. For $q = 6$ the estimated error goes up and the exact error goes down, the error now is overestimated by a factor of roughly 5. That we get smaller exact error means that the difference formulae of order 6 are still excellent, but the order 8 formulae that are used for the error estimate are overdrawn for this mesh and this would be detected by an order control that sees only the estimated errors. Note the strongly increased computation time because higher order is computationally more expensive.

In Table 2 we present the data for the second cycle if we allow pure order control. The results are quite similar for starting order (first cycle) $q = 2$ or 4. None or very few nodes have order $q = 6$. It is not surprising that the result is close to that of order 4 in Table 1. For $f$ the values mentioned in the context of Eq. (15) have been used.

Table 3 presents the results for pure mesh refinement with fixed order $q = 4$. We prescribe a relative tolerance $\text{tol} = 0.15 \cdot 10^{-2}$, i.e., 0.15% error which corresponds to 0.16 times the error of the basic solution. After the first cycle 218 nodes must be refined which produces $1584 - 751 = 833$ new nodes and $2997 - 1410 = 1587$ new elements because all elements that contain a refinement node are refined as can be seen in Fig. 1. In cycle 3 only 12 nodes must be refined. They produce 135 new nodes and 210 new elements. The final error estimate is excellent, Fig. 6 shows the 4 meshes for the 4 cycles of Table 3. In the fourth cycle only a few spots are refined. The spot at the boundary results probably from a bad choice of the nodes for the difference formulae.

Table 4 shows the refinement process like Table 3, but now we also allow the order control. In the second cycle the estimated error even increases and also the exact error is significantly larger than without order control. However, in the third cycle the errors drop significantly, here 38 nodes

Table 3
Pure mesh refinement with order $q = 4$ and $s_s = 24$ in (21), tol $= 0.15 \cdot 10^{-2}$ (independent total time 102.4 s)

| Cycle | No. of nodes | No. of elements | No. of nodes refined | Global relat. error | | Time for cycle (s) |
|---|---|---|---|---|---|---|
| | | | | Estimated | Exact | |
| 1 | 751 | 1410 | 218 | $0.970 \cdot 10^{-2}$ | $1.478 \cdot 10^{-2}$ | 9.4 |
| 2 | 1584 | 2997 | 243 | $0.204 \cdot 10^{-2}$ | $0.175 \cdot 10^{-2}$ | 22.3 |
| 3 | 2541 | 4818 | 12 | $0.167 \cdot 10^{-2}$ | $0.042 \cdot 10^{-2}$ | 31.9 |
| 4 | 2676 | 5028 | — | $0.024 \cdot 10^{-2}$ | $0.021 \cdot 10^{-2}$ | 38.7 |

Table 4
Mesh refinement and order control, tol $= 0.15 \cdot 10^{-2}$ (independent total time 83 s)

| Cycle | No. of nodes | No. of elements | No. of nodes refined | No. of nodes with order $q=$ | | | Global relat. error | | Time for cycle (s) |
|---|---|---|---|---|---|---|---|---|---|
| | | | | 2 | 4 | 6 | Estimated | Exact | |
| 1 | 751 | 1410 | 229 | 0 | 751 | 0 | $0.946 \cdot 10^{-2}$ | $1.478 \cdot 10^{-2}$ | 14.9 |
| 2 | 1597 | 3027 | 91 | 527 | 1068 | 2 | $1.037 \cdot 10^{-2}$ | $0.567 \cdot 10^{-2}$ | 28.2 |
| 3 | 2131 | 3963 | — | 262 | 1831 | 38 | $0.088 \cdot 10^{-2}$ | $0.099 \cdot 10^{-2}$ | 39.9 |

have the order $q = 6$. The exact error in the third cycle is larger than without order control, but with significantly less nodes and elements. Each cycle is more expensive than for fixed order, but as we need only 3 cycles, the total solution time is smaller. The final error estimate is again excellent. Fig. 7 shows the grids for the third cycle with fixed order $q = 4$ (Table 3) and with selfadapted order (Table 4). The refinements are visibly different. It should be noted that the requested accuracy is obtained for variable order with less nodes (and elements) than for fixed order. This may be of importance if the memory is the limiting factor.

We have presented only these examples based on the mesh of Fig. 3. We have applied the solution method to other configurations with other meshes with quite similar behavior so that these data would not give new information. The essential point is that the method is "intelligent" in the way it adapts itself to the problem to be solved. It should be recalled that the FDEM program package is a black-box solver and Eqs. (32), (33) are only one example that has been put into this black box.

It should be mentioned that in the FEM one would call the results of Tables 1 and 2 a $p$-method (change of order), of Table 3 a $h$-method (change of mesh size) and of Table 4 a $h$–$p$-method (change of mesh size and order).

One may ask: what is the cost of using the higher order difference methods as compared to the FEM? We consider as "cost" the computer time, e.g., the CPU time, to solve a certain problem for a prescribed relative error. If there is not built into the program a reliable error estimate this would mean at least two solutions on two different grids. The main computation time is usually the time for the linear solver. We have made a comparison of the FEM code VECFEM and the CADSOL code [12] (is the present method on a body-oriented grid where the indices determine

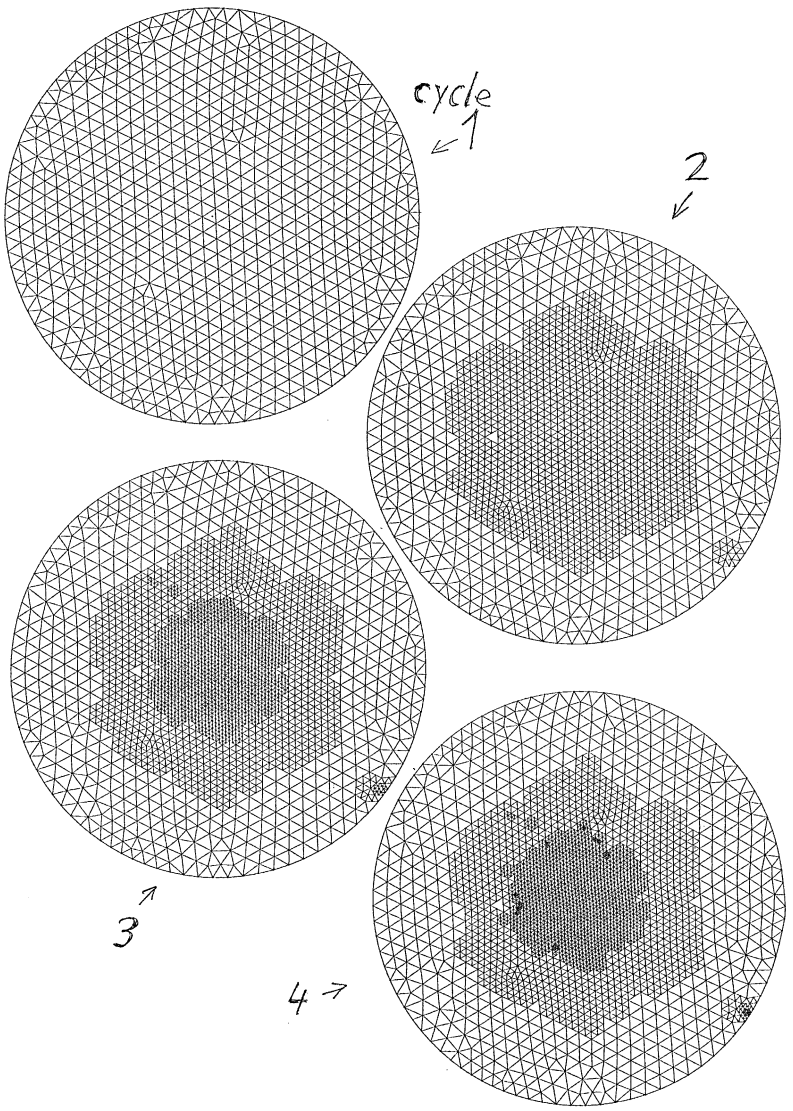


Fig. 6. The meshes for the 4 cycles of Table 3.

the neighbor), both using the same linear solver package LINSOL (essential for a comparison). We used a system of elliptic PDEs with known solution, a second-order FEM grid and method, and a fourth-order FDM on a CADSOL body-oriented grid so that equal accuracy was obtained. Under these conditions VECFEM was 20–30% faster, but with CADSOL we had a reliable error estimate immediately with the solution. As we have designed both codes ourselves, they are of equal quality (also an essential factor). This experiment cannot be repeated with FDEM because VECFEM uses an old version of LINSOL wheras FDEM uses an improved parallelized version of LINSOL.

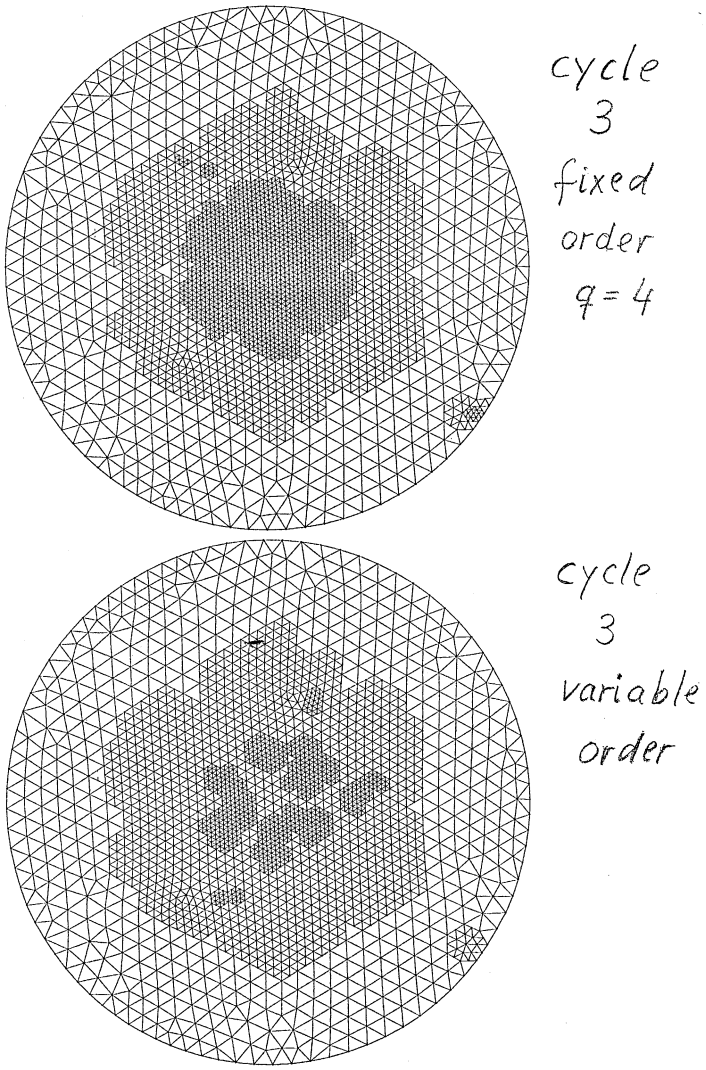


Fig. 7. Comparison of the meshes for the third cycle with fixed order $q = 4$ (above) and with self-adapted order (below).

One may ask: what is the cost of the computation of the difference formaulae which is ultimately the cost for the inversion of the matrix $M$, $A = M^{-1}$, Eq. (28) by the Gauss–Jordan algorithm? We consider Table 4 with order and mesh size control. The total solution time is 83 s. In each of the three cycles we compute a difference formula and an error formula for the orders $2, 4, 6$ for each node $(751, 1597, 2131)$. This costs totally 49.5 s, i.e., roughly 60% of the total time. However, this percentage results from the fact that the linear system converges very fast. Nevertheless, the computation of the difference and error formulae will always be a significant part of the solution time and is the price for the immediate quality control. If we computed only difference formulae (no error formulae), we would need, e.g., for the order 4 roughly 15% of the corresponding Gauss–Jordan

time because the errors for the order 4 (15 nodes) are estimated by the formulae of order 6 (28 nodes) and we must compare $15^3 = 3375$ to $28^3 = 21\,952$ for the computation by Gauss–Jordan. So we see that the computation of the error formulae is dominant: the error estimates are not free.

As a consequence of these considerations we developed a "lean" version of our program that computes the solution with order 2 on a fixed mesh without error control, with reduction of the Newton residual by a factor $10^2$. Such a procedure is comparable to the usual FEM packages. We recomputed with the lean version the case $q = 2$ of Table 1 that needed with all control parts 5.7 s. The lean version needed 1.3 s. The difference are the costs for the control of the Newton iteration and for the error estimate. So a user can for coarse trial computation choose the "lean" option and then do only the final computation with error estimates and eventual order and mesh size control.

## 7. A remark to the linear solver LINSOL

The solution of the large and sparse linear system with the matrix $Q_d$, e.g., (8) for the Newton correction or (6) for the total error, is computed with our LINSOL program package [9,10,14]. LINSOL is not the subject of this paper. Nevertheless it may be useful to note here some properties of LINSOL (for details see the references). It is fully parallelized, with parallelized bandwidth optimizer and (I)LU preconditioner. It has presently 14 generalized CG methods from which several polyalgorithms with automatic method selection are generated. The matrix is composed from "elementary" matrices with 10 different data structures, e.g., full and packed diagonals, rows, columns or starry sky (double index). The stopping criterion for LINSOL, see Chapter 17 in [13], considers that no digits are computed that are overwritten by the next Newton correction or that are below the required accuracy or below the discretization error. Thus there is a sophisticated interplay between LINSOL and FDEM.

## 8. Concluding remarks

For many years we have worked in our group "Numerical Research for Supercomputers" in the field of solution methods for elliptic and parabolic PDEs. We have developed the FIDISOL program package, see [13, Section 17] where we learned to generate difference and error formulae on a rectangular grid and to use this information for order and step size control. We have developed the CADSOL program package [12] where we learned how to generate difference and error formulae on an arbitrary but body-oriented grid. Then we developed the VECFEM program package [7], a black-box FEM solver. Here we learned how to deal with unstructured FEM meshes – and how difficult it is to get for such a FEM black-box an error estimate [8]. An order control in the above presented way is not at all possible for the FEM. In the FDEM program package we now combine all these experiences to an "ideal" PDE black-box solver.

As mentioned above our solution method is based on the error Eq. (6). It is the knowledge of the error that makes the method "intelligent" and allows the selection of the optimal order and mesh size. Parallel to these discretization method the development of an "intelligent" iterative linear solver is a necessity. We have developed the LINSOL program package [9,10,14] with polyalgorithms that

automatically select the optimal (CG-type) method by a polyalgorithm out of a given "scale" of methods.

This paper is a report about work in progress. The next steps are the extension of the FDEM to 3-D (this has been started), then we add the time step size and order control of FIDISOL to get the parabolic solver (this is a pure matter of diligence). Then the program package will be parallelized, using the vast amount of experience that we have gained over many years, see [16]. If the package has been finished it will be available as public domain software with the source code, so everybody will be able to "play" with it. Finally, we will design a symbolic user-interface so that the user has to enter his PDEs and BCs in symbolic form and the Fortran code and the Jacobian matrices with their code are generated automatically by a computer algebra program, e.g., MAPLE.

Questions that have not been addressed in this paper are mesh coarsening and moving grids. These problems will be treated together with the parabolic solver. Then we will also introduce "dividing lines" like in CADSOL [12] that allow a jump in the PDEs and BCs so that it is not possible to differentiate over such lines and there the PDEs are replaced by coupling conditions.

## Acknowledgements

We thank an anonymous referee for valuable remarks. His hints have essentially contributed to improve the legibility of this paper.

## References

[1] R. Abgrall, S. Lanteri, T. Sonar, ENO approximations for compressible fluid dynamics, Z. Angew. Math. Mech. 79 (Suppl. 1) (1999) 3–28.
[2] R.E. Bank, PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, SIAM, Philadelphia, 1994.
[3] P. Bastian et al., A parallel software platform for solving problems of partial differential equations using unstructured grids and adaptive multigrid methods, in: E. Krause, W. Jäger (Eds.), High Performance Computing in Science and Engineering '98, Springer, Berlin, 1999, pp. 326–339.
[4] J.G. Blom, R.A. Trompert, J.G. Verwer, VLUGR2. a vectorizable adaptive grid solver for PDEs in 2 D, Report NM-R9403, CWI, Amsterdam, 1994.
[5] J.G. Blom, J.G. Verwer, VLUGR3: A vectorizable adaptive grid solver for PDEs in 3 D. I. Algorithmic aspects and applications, Appl. Numer. Math. 16 (1994) 129–156 and CWI Report NM-R9404.
[6] J.G. Blom, J.G. Verwer, VLUGR3: A vectorizable adaptive grid solver for PDEs in 3D. II. Code description, Report NM-R9405, CWI, Amsterdam, 1994.
[7] L. Grosz, C. Roll, W. Schönauer, A black box solver for the numerical solution of general nonlinear functional equations by mixed FEM, in: M. Krizek, P. Nettaanmaeki, R. Steenberg (Eds.), Fifty Years of the Courant Element, Marcel Dekker, New York 1994, pp. 225–234. This paper and further documents are available under the URL: http://www.rz.uni-karlsruhe.de/Uni/RZ/Forschung/Numerik/vecfem/docs.html.
[8] L. Grosz, A-posteriori error estimates for the finite element solution of non-linear variational problems, Dissertation Universität Karlsruhe, 1997, this dissertation is available under the URL given in [1].
[9] H. Häfner, W. Schönauer, R. Weiss, The parallel and portable linear solver package LINSOL, in: H. Lederer, F. Hertweck (Eds.), Proceedings of the Fourth European SGI/Cray MPP Workshop, Max-Planck-Institut für Plasmaphysik, Garching bei München, Germany, 1998, pp. 242–251.
[10] H. Häfner, W. Schönauere, R. Weiss, Parallelization and integration of the LU and ILL algorithm in the LINSOL program package, in: V. Malyskin (Ed.), Parallel Computing Technologies, Springer Lecture Notes in Computer Science, Vol. 1662, Springer, Berlin, 1999, pp. 417–427.

[11] H. Rannacher, Error control in finite element computations, in: C. Bulgak, C. Zenger (Eds.), Error Control and Adaptivity in Scientific Computing, Kluwer Academic Publishers, Dordrecht, Netherlands, 1999, pp. 247–278.
[12] M. Schmauder, W. Schönauer, CADSOL — A fully vectorized black box solver for 2-D and 3-D partial differential equations, in: R. Vichnevetsky, D. Knight, G. Richter (Eds.), Advances in Computer Methods for Partial Differential Equations — VII, IMACS, New Brunswick, 1992, pp. 639–645.
[13] W. Schönauer, Scientific Computing on Vector Computers, North-Holland, Amsterdam, 1987.
[14] W. Schönauer, H. Häfner, R. Weiss, LINSOL, a parallel iterative linear solver package of generalized CG-type for sparse matrices, in: M. Heath et al. (Eds.), Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, SIAM, Philadelphia 1997, CD-ROM (ISBN 0-89871-395-1), 8 pp.
[15] W. Schönauer, Generation of difference and error formulae of arbitrary consistency order on an unstructured grid, to appear in Z. Angew. Math. Mech. 78 (Suppl. 3) (1998) 1061–1062 (special issue of the GAMM-Tagung 1997).
[16] W. Schönauer, Scientific supercomputing, available in the internet by the URL: http://www.uni-karlsruhe.de/Uni/RZ/Personen/rz03/book/.